**cyberbullying\_detection\_nlp.py**

import os

from tkinter import \*

import tkinter

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

from sklearn.feature\_extraction.text import CountVectorizer

import nltk

import numpy as np

import pandas as pd

import seaborn as sns

from tqdm.auto import tqdm

from nltk import PorterStemmer

from nltk.corpus import stopwords

from sklearn import \*

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_curve, auc,confusion\_matrix

main = tkinter.Tk()

main.title("Automated Emerging Cyber Threat Identification and Profiting based on Natural Language Processing")

main.geometry("1300x1200")

global filename

global balance\_data

global data

global X, Y, X\_train, X\_test, y\_train, y\_test

global LR\_acc, NB\_acc, RFT\_acc,Xgb\_acc,DT\_acc

def upload():

global filename

text.delete('1.0', END)

filename = askopenfilename(initialdir="Dataset")

pathlabel.config(text=filename)

text.insert(END, "Dataset loaded\n\n")

def preprocess():

global filenam

text.delete('1.0', END)

balance\_data = pd.read\_csv(filename, nrows = 5000)

text.insert(END, "Information about the dataset\n\n")

text.insert(END, balance\_data.head())

text.insert(END, "\n\n")

# Shape of dataframe

text.insert(END, "Shape of dataframe\n\n")

text.insert(END, balance\_data.shape)

text.insert(END, "\n\n")

# Listing the features of the dataset

text.insert(END, "Listing the features of the dataset\n\n")

text.insert(END, balance\_data.columns)

text.insert(END, "\n\n")

# Information about the dataset

balance\_data.info()

# 1. Handling Null Values

balance\_data.isna().any()

balance\_data.isna().sum()

"""### 2. Handling Duplicate Values"""

balance\_data.nunique()

balance\_data['tweet\_text'].nunique()

"""### 3. Class Distributions"""

balance\_data['cyberbullying\_type'].value\_counts()

# Create a bar plot of the class distribution

class\_counts = balance\_data['cyberbullying\_type'].value\_counts()

class\_counts.plot(kind='bar')

plt.title('Class Distribution of Cyberbullying Types')

plt.xlabel('Labels')

plt.ylabel('Number of Tweets')

plt.show()

"""### 4. Word Count"""

from collections import Counter

import re

import nltk

from nltk.corpus import stopwords

# Concatenate all tweet texts into a single string

all\_text = ' '.join(balance\_data['tweet\_text'].values)

# Remove URLs, mentions, and hashtags from the text

all\_text = re.sub(r'http\S+', '', all\_text)

all\_text = re.sub(r'@\S+', '', all\_text)

all\_text = re.sub(r'#\S+', '', all\_text)

# Split the text into individual words

words = all\_text.split()

# Remove stop words

stop\_words = set(stopwords.words('english'))

words = [word for word in words if not word in stop\_words]

text.insert(END, "\n")

text.insert(END, "Count the frequency of each word")

# Count the frequency of each word

word\_counts = Counter(words)

top\_words = word\_counts.most\_common(100)

text.insert(END,"\n")

text.insert(END, top\_words)

text.insert(END, "\n")

# Create a bar chart of the most common words

top\_words = word\_counts.most\_common(10) # Change the number to show more/less words

x\_values = [word[0] for word in top\_words]

y\_values = [word[1] for word in top\_words]

plt.bar(x\_values, y\_values)

plt.xlabel('Word')

plt.ylabel('Frequency')

plt.title('Most Commonly Used Words')

plt.show()

text.insert(END, "Removed non numeric characters from dataset\n\n")

def SentimentAnalysis():

text.delete('1.0', END)

import pandas as pd

import matplotlib.pyplot as plt

from textblob import TextBlob

global balance\_data

balance\_data = pd.read\_csv(filename, nrows = 5000)

# perform sentiment analysis on each text in DataFrame

sentiment\_scores = []

for text1 in balance\_data['tweet\_text']:

analysis = TextBlob(text1)

sentiment\_scores.append((analysis.sentiment.polarity, analysis.sentiment.subjectivity))

text.insert(END, "\n")

text.insert(END, "sentiment scores for all tweets")

# create DataFrame with sentiment scores

sentiment\_df = pd.DataFrame(sentiment\_scores, columns=['polarity', 'subjectivity'])

text.insert(END, "\n")

text.insert(END, sentiment\_df)

# plot distribution of sentiment scores

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))

sentiment\_df['polarity'].plot(kind='hist', ax=axes[0], title='Polarity')

sentiment\_df['subjectivity'].plot(kind='hist', ax=axes[1], title='Subjectivity')

plt.show()

def nerModel():

text.delete('1.0', END)

global balance\_data

import spacy

from spacy import displacy

# sample text

text1 = balance\_data['tweet\_text'].iloc[4]

# load pre-trained NER model

nlp = spacy.load('en\_core\_web\_sm')

# perform named entity recognition

doc = nlp(text1)

# visualize named entities

displacy.render(doc, style='ent', jupyter=True)

text.insert(END, "perform named entity recognition")

text.insert(END, "\n")

text.insert(END, doc)

text.insert(END, "\n")

def posModel():

import spacy

from spacy import displacy

text.delete('1.0', END)

global balance\_data

# sample text

text1 = balance\_data['tweet\_text'].iloc[1]

# load pre-trained POS tagging model

nlp = spacy.load('en\_core\_web\_sm')

# perform POS tagging

doc = nlp(text1)

# visualize POS tagging

displacy.render(doc, style='dep', jupyter=True, options={'distance': 90})

text.insert(END, "\n")

text.insert(END, "perform POS tagging")

text.insert(END, "\n")

text.insert(END, doc)

def topicModel():

text.delete('1.0', END)

global balance\_data

global X, Y, X\_train, X\_test, y\_train, y\_test

import gensim

# Preprocessing

tokens = [[word for word in sentence.split()] for sentence in balance\_data['tweet\_text']]

dictionary = gensim.corpora.Dictionary(tokens)

corpus = [dictionary.doc2bow(token) for token in tokens]

# Topic Modeling

num\_topics = 10

lda\_model = gensim.models.LdaModel(corpus=corpus, id2word=dictionary, num\_topics=num\_topics)

text.insert(END, "\n")

text.insert(END, "Data Cleaning Process is started...")

def clean\_text(texta):

# Remove HTML tags

texta = re.sub('<.\*?>', '', texta)

# Remove non-alphabetic characters and convert to lowercase

texta = re.sub('[^a-zA-Z]', ' ', texta).lower()

# Remove URLs, mentions, and hashtags from the text

texta = re.sub(r'http\S+', '', texta)

texta = re.sub(r'@\S+', '', texta)

texta = re.sub(r'#\S+', '', texta)

# Tokenize the text

words = nltk.word\_tokenize(texta)

# Remove stopwords

words = [w for w in words if w not in stopwords.words('english')]

# Stem the words

stemmer = PorterStemmer()

words = [stemmer.stem(w) for w in words]

# Join the words back into a string

texta = ' '.join(words)

text.insert(END, "\n")

text.insert(END, texta)

return texta

tqdm.pandas()

balance\_data['cleaned\_text'] = balance\_data['tweet\_text'].progress\_apply(clean\_text)

# Create the Bag of Words model

cv = CountVectorizer()

X = cv.fit\_transform(balance\_data['cleaned\_text']).toarray()

Y = balance\_data['cyberbullying\_type']

def generateModel():

text.delete('1.0', END)

global balance\_data

global X,Y,X\_train, X\_test, y\_train, y\_test

"""## 4. Splitting the Data:

The data is split into train & test sets, 80-20 split.

"""

# Splitting the dataset into train and test sets: 80-20 split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

text.insert(END, "Train & Test Model Generated\n\n")

text.insert(END, "Total Dataset Size : " + str(len(balance\_data)) + "\n")

text.insert(END, "Split Training Size : " + str(len(X\_train)) + "\n")

text.insert(END, "Split Test Size : " + str(len(X\_test)) + "\n")

def plot\_confusion\_matrix(test\_Y, predict\_y):

C = confusion\_matrix(test\_Y, predict\_y)

A =(((C.T)/(C.sum(axis=1))).T)

B =(C/C.sum(axis=0))

plt.figure(figsize=(20,4))

labels = [1,2]

cmap=sns.light\_palette("blue")

plt.subplot(1, 3, 1)

sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Original Class')

plt.title("Confusion matrix")

plt.subplot(1, 3, 2)

sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Original Class')

plt.title("Precision matrix")

plt.subplot(1, 3, 3)

sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted Class')

plt.ylabel('Original Class')

plt.title("Recall matrix")

plt.show()

def runRFT():

text.delete('1.0', END)

global RFT\_acc

global classifier

global X, Y, X\_train, X\_test, y\_train, y\_test

total = X\_train.shape[1];

text.insert(END, "Total Features : " + str(total) + "\n")

from sklearn.ensemble import RandomForestClassifier

# instantiate the model

forest = RandomForestClassifier(n\_estimators=10)

# fit the model

forest.fit(X\_train, y\_train)

# predicting the target value from the model for the samples

y\_train\_forest = forest.predict(X\_train)

y\_test\_forest = forest.predict(X\_test)

# computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_forest = metrics.accuracy\_score(y\_train, y\_train\_forest)

acc\_test\_forest = metrics.accuracy\_score(y\_test, y\_test\_forest)

RFT\_acc = acc\_test\_forest

# computing the accuracy, f1\_score, Recall, precision of the model performance

text.insert(END, "\n")

text.insert(END, "Random Forest : Accuracy on training Data: {:.3f}".format(acc\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Random Forest : Accuracy on test Data: {:.3f}".format(acc\_test\_forest))

text.insert(END, "\n")

f1\_score\_train\_forest = metrics.f1\_score(y\_train, y\_train\_forest)

f1\_score\_test\_forest = metrics.f1\_score(y\_test, y\_test\_forest)

text.insert(END, "\n")

text.insert(END, "Random Forest : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Random Forest : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_forest))

text.insert(END, "\n")

recall\_score\_train\_forest = metrics.recall\_score(y\_train, y\_train\_forest)

recall\_score\_test\_forest = metrics.recall\_score(y\_test, y\_test\_forest)

text.insert(END, "\n")

text.insert(END, "Random Forest : Recall on training Data: {:.3f}".format(recall\_score\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Random Forest : Recall on test Data: {:.3f}".format(recall\_score\_test\_forest))

text.insert(END, "\n")

precision\_score\_train\_gbc = metrics.precision\_score(y\_train, y\_train\_forest)

precision\_score\_test\_gbc = metrics.precision\_score(y\_test, y\_test\_forest)

text.insert(END, "\n")

text.insert(END, "Random Forest : precision on training Data: {:.3f}".format(precision\_score\_train\_gbc))

text.insert(END, "\n")

text.insert(END, "Random Forest : precision on test Data: {:.3f}".format(precision\_score\_test\_gbc))

text.insert(END, "\n")

# computing the classification report of the model

text.insert(END, "\n")

text.insert(END, "Classification Report: \n")

text.insert(END, metrics.classification\_report(y\_test, y\_test\_forest))

plot\_confusion\_matrix(y\_train, y\_train\_forest)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_forest)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkgreen', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='lightgreen', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.grid(False)

plt.show()

def runLR():

text.delete('1.0', END)

global LR\_acc

global classifier

global X, Y, X\_train, X\_test, y\_train, y\_test

total = X\_train.shape[1];

text.insert(END, "Total Features : " + str(total) + "\n")

from sklearn.linear\_model import LogisticRegression

# from sklearn.pipeline import Pipeline

# instantiate the model

log = LogisticRegression(max\_iter = 1000)

# fit the model

log.fit(X\_train, y\_train)

# predicting the target value from the model for the samples

y\_train\_log = log.predict(X\_train)

y\_test\_log = log.predict(X\_test)

# computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_svc = metrics.accuracy\_score(y\_train, y\_train\_log)

acc\_test\_svc = metrics.accuracy\_score(y\_test, y\_test\_log)

LR\_acc=acc\_test\_svc

# computing the accuracy, f1\_score, Recall, precision of the model performance

text.insert(END, "\n")

text.insert(END, "Logistic Regression : Accuracy on training Data: {:.3f}".format(acc\_train\_svc))

text.insert(END, "\n")

text.insert(END, "Logistic Regression : Accuracy on test Data: {:.3f}".format(acc\_test\_svc))

text.insert(END, "\n")

f1\_score\_train\_svc = metrics.f1\_score(y\_train, y\_train\_log)

f1\_score\_test\_svc = metrics.f1\_score(y\_test, y\_test\_log)

text.insert(END, "\n")

text.insert(END, "Logistic Regression : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_svc))

text.insert(END, "\n")

text.insert(END, "Logistic Regression : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_svc))

text.insert(END, "\n")

recall\_score\_train\_gbc = metrics.recall\_score(y\_train, y\_train\_log)

recall\_score\_test\_gbc = metrics.recall\_score(y\_test, y\_test\_log)

text.insert(END, "\n")

text.insert(END, "Logistic Regression : Recall on training Data: {:.3f}".format(recall\_score\_train\_gbc))

text.insert(END, "\n")

text.insert(END, "Logistic Regression : Recall on test Data: {:.3f}".format(recall\_score\_test\_gbc))

text.insert(END, "\n")

precision\_score\_train\_gbc = metrics.precision\_score(y\_train, y\_train\_log)

precision\_score\_test\_gbc = metrics.precision\_score(y\_test, y\_test\_log)

text.insert(END, "\n")

text.insert(END, "Logistic Regression : precision on training Data: {:.3f}".format(precision\_score\_train\_gbc))

text.insert(END, "\n")

text.insert(END, "Logistic Regression : precision on test Data: {:.3f}".format(precision\_score\_test\_gbc))

text.insert(END, "\n")

# computing the classification report of the model

text.insert(END, "\n")

text.insert(END, "Classification Report: \n")

text.insert(END, metrics.classification\_report(y\_test, y\_test\_log))

plot\_confusion\_matrix(y\_train, y\_train\_log)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_log)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkgreen', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='lightgreen', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.grid(False)

plt.show()

def runDT():

text.delete('1.0', END)

global DT\_acc

global classifier

global X, Y, X\_train, X\_test, y\_train, y\_test

total = X\_train.shape[1];

text.insert(END, "Total Features : " + str(total) + "\n")

from sklearn.tree import DecisionTreeClassifier

# instantiate the model

tree = DecisionTreeClassifier(max\_depth=30)

# fit the model

tree.fit(X\_train, y\_train)

# predicting the target value from the model for the samples

y\_train\_tree = tree.predict(X\_train)

y\_test\_tree = tree.predict(X\_test)

# computing the accuracy, f1\_score, Recall, precision of the model performance

acc\_train\_forest = metrics.accuracy\_score(y\_train, y\_train\_tree)

acc\_test\_forest = metrics.accuracy\_score(y\_test, y\_test\_tree)

DT\_acc=acc\_test\_forest

# computing the accuracy, f1\_score, Recall, precision of the model performance

text.insert(END, "\n")

text.insert(END, "Decision Tree : Accuracy on training Data: {:.3f}".format(acc\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Decision Tree : Accuracy on test Data: {:.3f}".format(acc\_test\_forest))

text.insert(END, "\n")

f1\_score\_train\_forest = metrics.f1\_score(y\_train, y\_train\_tree)

f1\_score\_test\_forest = metrics.f1\_score(y\_test, y\_test\_tree)

text.insert(END, "\n")

text.insert(END, "Decision Tree : f1\_score on training Data: {:.3f}".format(f1\_score\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Decision Tree : f1\_score on test Data: {:.3f}".format(f1\_score\_test\_forest))

text.insert(END, "\n")

recall\_score\_train\_forest = metrics.recall\_score(y\_train, y\_train\_tree)

recall\_score\_test\_forest = metrics.recall\_score(y\_test, y\_test\_tree)

text.insert(END, "\n")

text.insert(END, "Decision Tree : Recall on training Data: {:.3f}".format(recall\_score\_train\_forest))

text.insert(END, "\n")

text.insert(END, "Decision Tree : Recall on test Data: {:.3f}".format(recall\_score\_test\_forest))

text.insert(END, "\n")

precision\_score\_train\_gbc = metrics.precision\_score(y\_train, y\_train\_tree)

precision\_score\_test\_gbc = metrics.precision\_score(y\_test, y\_test\_tree)

text.insert(END, "\n")

text.insert(END, "Decision Tree : precision on training Data: {:.3f}".format(precision\_score\_train\_gbc))

text.insert(END, "\n")

text.insert(END, "Decision Tree : precision on test Data: {:.3f}".format(precision\_score\_test\_gbc))

text.insert(END, "\n")

# computing the classification report of the model

text.insert(END, "\n")

text.insert(END, "Classification Report: \n")

text.insert(END, metrics.classification\_report(y\_test, y\_test\_tree))

plot\_confusion\_matrix(y\_train, y\_train\_tree)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_tree)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkgreen', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='lightgreen', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc='lower right')

plt.grid(False)

plt.show()

def detectAttack():

text.delete('1.0', END)

# then make a url variable

url = "http://127.0.0.1:5000"

# then call the get method to select the code

# for new browser and call open method

# described above

import webbrowser

webbrowser.open(url, new=0, autoraise=True)

os.system('python app.py')

def graph():

height = [LR\_acc,RFT\_acc,DT\_acc]

bars = ('LR Accuracy','RFT Accuracy','DT Accuracy')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.xlabel("Explainable AI Techniques")

plt.ylabel("Accuracy Score")

plt.title("Comparison of Performance Estimation")

plt.show()

font = ('times', 14, 'bold')

title = Label(main,

text='AUTOMATED EMERGING CYBER THREAT INDENTICATION AND PROFITING BASED ON NATURAL LANGUAGE PROCESSING')

title.config(bg='pink', fg='brown')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0, y=0)

font1 = ('times', 14, 'bold')

upload = Button(main, text="Upload Cyberbullying Tweets Dataset", command=upload)

upload.place(x=700, y=100)

upload.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='DarkOrange1', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=700, y=150)

preprocess = Button(main, text="Dataset Pre-processing", command=preprocess)

preprocess.place(x=700, y=200)

preprocess.config(font=font1)

SentimentAna = Button(main, text="Sentiment Analysis", command=SentimentAnalysis)

SentimentAna.place(x=700, y=250)

SentimentAna.config(font=font1)

NER= Button(main, text="Named Entity Recognition (NER)", command=nerModel)

NER.place(x=700, y=300)

NER.config(font=font1)

pos= Button(main, text="Part-of-Speech (POS) Tagging", command=posModel)

pos.place(x=700, y=350)

pos.config(font=font1)

TopicModeling= Button(main, text="Topic Modeling Visualization", command=topicModel)

TopicModeling.place(x=700, y=400)

TopicModeling.config(font=font1)

model = Button(main, text="Feature Extraction", command=generateModel)

model.place(x=700, y=450)

model.config(font=font1)

runsvm = Button(main, text="Logistics Regression Algorithm", command=runLR)

runsvm.place(x=700, y=500)

runsvm.config(font=font1)

rundet = Button(main, text="Decision Tree Classifier Algorithm", command=runDT)

rundet.place(x=700, y=550)

rundet.config(font=font1)

runsvm = Button(main, text="Random Forest Algorithm", command=runRFT)

runsvm.place(x=700, y=600)

runsvm.config(font=font1)

graphButton = Button(main, text="Comparision of Models", command=graph)

graphButton.place(x=700, y=650)

graphButton.config(font=font1)

font1 = ('times', 12, 'bold')

text = Text(main, height=30, width=80)

scroll = Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10, y=100)

text.config(font=font1)

main.config(bg='brown')

main.mainloop()